Audio Programming 2

Peter Dowsett

peter.dowsett@uwl.ac.uk

Audio Programming 2

Audio Plug-in Frameworks

Writing Audio Discussion

- Possible to compute sample values from scratch and write them to file => synthesis
- Making your code robust against misuse by validating input

  Note: A little additional effort spent now will pay itself back in the future

Audio Plug-in Frameworks

Here is where it starts to get “real”.

- Not much more difficult than what we’ve seen so far
- In some ways easier: memory management etc. taken care of by framework
- Can come with complex “ecosystem” of graphics libraries, tools and software engineering concepts, but the basics are not complex

Audio Plug-In Framework Benefits

Audio plug-in frameworks make it easier to create audio products by handling some of the more “low level tasks” for you.

These can be but aren’t limited to:

- Operating System needs
- The interaction with a host DAW
- Handling the nuances of different plug-in formats (AAX, AU, VST etc)
- Having an engine for customisable graphics

The Audio Callback Function

When dealing with real-time audio there is a potentially endless stream of audio frames chopped into slices, called buffers.

- The chosen Framework takes care of scheduling, presents slices at regular(-ish) intervals
- “We” programmers only need to worry about processing incoming buffers and filling the associated output buffers

Audio Callback Examples

VST3: tresult IAudioProcessor:process (ProcessData &data)=0

JUCE: void AudioProcessor::processBlock (AudioSampleBuffer& buffer, MidiBuffer& midiMessages)

iPlug2: void Plugin::ProcessBlock(sample** inputs, sample** outputs, int nFrames)

LV2: static void run(LV2_Handle instance, uint32_t n_samples)

Bela: void render(BelaContext* context, void* userData)

Not just DSP

As well as having to deal with Digital Signal Processing users of audio frameworks will typically have:

- **Parameters** – These are the user controllable variables which change the operation of the DSP.

  Question: How might these be tied to the UI?

- **Presets** – How can the parameters settings be loaded and saved?
- **Graphical User Interface** – How does the user interact with the parameters?

  Note: A custom UI is not strictly necessary and UI can default to generic interface based on parameter descriptions.

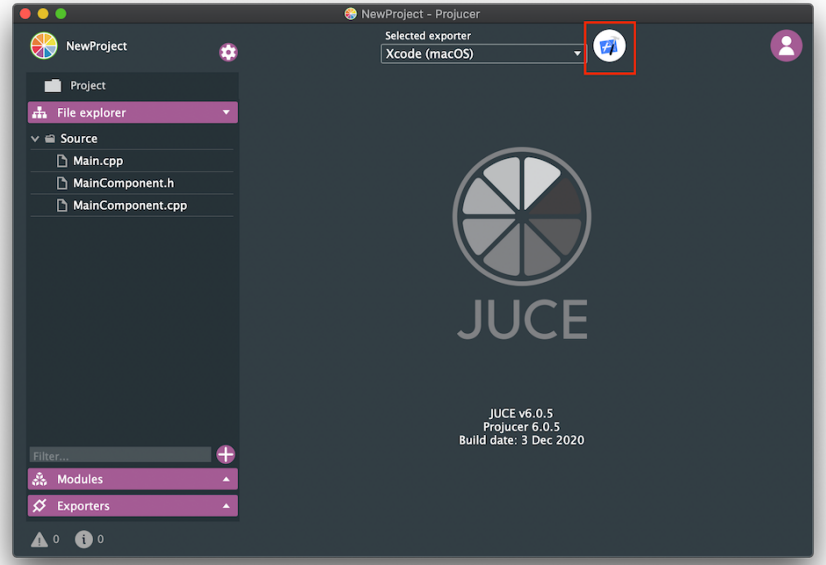
Your First JUCE Plug-in

Exercise:

Create an audio plug-in which has a single parameter for audio gain.

This should range between a minimum of -48 dB and a maximum of +12 dB, an increment size of 0.1dB and a default value of unity gain (0dB).

Do not spent time writing a completely custom UI, only use the base functionality of the juce::Slider class and basic juce::Graphics object functions.



The Projucer

The Projucer is the place where you add all of your plug-ins main settings that need to be replicated across platforms.

In other words, if you need to make a Windows version of your plug-in, the Projucer can create a Visual Studio project for you based on your Projucer settings.

In this regard the Projucer is like a graphical version of cmake or other build tool.

Tutorials

The Projucer:
https://docs.juce.com/master/tutorial_new_projucer_project.html

https://docs.juce.com/master/tutorial_manage_projucer_project.html

Creating a basic audio plug-in:

https://docs.juce.com/master/tutorial_create_projucer_basic_plugin.html

https://docs.juce.com/master/tutorial_code_basic_plugin.html